```
namespace NeuralLibrary
{
    /// <summary>
       /// The connection held between two neurons with a
       given weight.
    public class Connection
    {
        /// <summary>
        /// Initializes the connection.
        public Connection(double weightInitial, Neuron
anteriorNeuron,                        Neuron posteriorNeuron)
        {
            this.Weight = weightInitial;
            this.AnteriorNeuron = anteriorNeuron;
            this.PosteriorNeuron = posteriorNeuron;
        }
        /// <summary>
        /// Initializes the connection with a random weight
between -1 and 1
        public Connection(Neuron anteriorNeuron, Neuron
posteriorNeuron)
            : this(Gaussian.GetRandomGaussian(),
anteriorNeuron, posteriorNeuron){}
        /// <summary>
        /// Nudges the weights.
        public void NudgeWeight()
        {
            this.Weight = Gaussian.GetRandomGaussian();
        }
        /// <summary>
        /// Feeds the product of output from the anterior
neuron  and the weight of the connection forward to the
anterior neuron.
        public void FeedForward()
        {
            PosteriorNeuron.Net += AnteriorNeuron.Output *
Weight;
        }

        #region Fields
        /// <summary>
        /// The last delta weight (used for momentum)
        protected double lastDeltaWeight = 0;
        protected double lastGradient = 0;
        protected double velocity = 0;

        #endregion Fields

        #region Properties
        /// <summary>
        /// The anterior neuron within the connection.
        public Neuron AnteriorNeuron { protected set; get; }
        /// <summary>
        /// The posterior neuron within the connection.
        public Neuron PosteriorNeuron { protected set; get; }
        /// <summary>
        /// Updates weight using the weight update rule. dW =
ERROR_posterior * OUTPUT_anterior
        public virtual void UpdateWeight(double
accelerationConstant, double momentum)
        {
            if (lastGradient * Gradient > 0)
                velocity += accelerationConstant;
            else if (lastGradient * Gradient < 0)
                velocity = 0;
            else
                velocity = accelerationConstant;

        double deltaWeight = -(Gradient*velocity) + momentum *
        lastDeltaWeight;
            Weight += deltaWeight;
            lastDeltaWeight = deltaWeight;
            lastGradient = Gradient;
        }
        /// <summary>
        /// Gets the gradient of the connection,
        public double Gradient
        {
            get
            {
                double output = 0;
                if (AnteriorNeuron is BiasNeuron)
                    output = (AnteriorNeuron as
BiasNeuron).Output;
                else if (AnteriorNeuron is InputNeuron)
                    output = (AnteriorNeuron as
InputNeuron).Output;
                else
                    output = AnteriorNeuron.Output;

                return PosteriorNeuron.Error * output;
            }
        }
        /// <summary>
        /// The weight associated with a connection.
        public double Weight { set; get; }

        #endregion Properties
    }
}
```